Detecting concurrency errors in multi-thr aded programs

Patent Number: EP0864975

Publication date: 1998-09-16

Inventor(s): SAVAGE STEFAN (US); BURROWS MICHAEL (US); NELSON CHARLES G (US); SOBALVARRO PATRICK G (US)

Applicant(s):: DIGITAL EQUIPMENT CORP (US)

Requested Patent: JP10254716

Application Number: EP19980104160 19980309

Priority Number(s): US19970815979 19970310

IPC Classification: G06F11/00

EC Classification: G06F11/00A4

Equivalents: CA2231597, US6009269

Abstract

A computer implemented method detects concurrency errors in programs. Machine executable images of multiple program threads are instrumented to locate and replace instructions which effect concurrency states of the threads. Concurrency state information is recorded in a memory while the multiple threads are executing. The recorded concurrency state information is analyzed, and inconsistent dynamic concurrency state transitions are reported as concurrency errors.

(19)日本国特許庁（ＪＰ）　(12) 公開特許公報（A）　(11)特許出願公開番号

特開平10－254716

(43)公開日　平成10年(1998)９月25日

(51)Int.Cl.[6]	識別記号	ＦＩ	
Ｇ０６Ｆ　9/46	３４０	Ｇ０６Ｆ　9/46	３４０Ｇ
11/30	３０５	11/30	３０５Ｄ
11/34		11/34	Ａ

審査請求　未請求　請求項の数10　ＯＬ　（全 12 頁）

(21)出願番号　特願平10－58064

(22)出願日　平成10年(1998)３月10日

(31)優先権主張番号　０８／８１５９７９

(32)優先日　1997年３月10日

(33)優先権主張国　米国（ＵＳ）

(71)出願人　590002873
ディジタル　イクイプメント　コーポレイション
アメリカ合衆国　マサチューセッツ州　01754－1418　メイナード　パウダー　ミル　ロード　111

(72)発明者　マイケル　バローズ
アメリカ合衆国　カリフォルニア州　94303　パロ　アルト　ディヴィッド　コート　3143

(74)代理人　弁理士　中村　稔　（外６名）

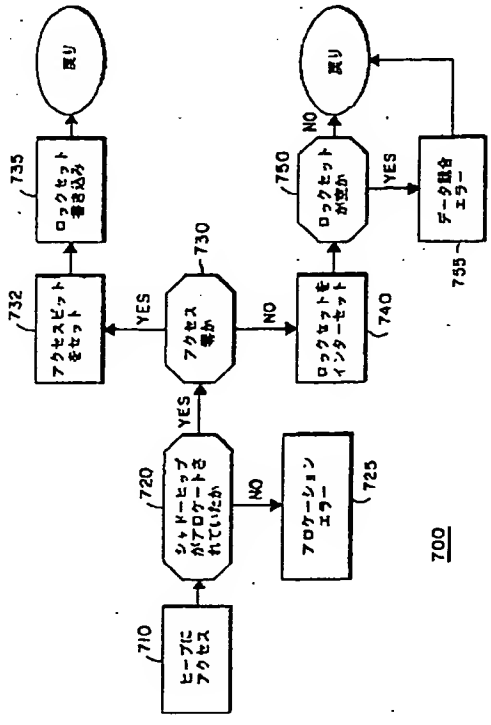
最終頁に続く

(54)【発明の名称】　マルチスレッデッドプログラムにおけるコンカレントエラーの検出

(57)【要約】

【課題】　プログラムスレッドをコンカレントに実行するに際し、コンカレントエラーを検出すること。

【解決手段】　コンピュータによって実施される方法は、プログラムにおけるコンカレントエラーを検出する。複数のプログラムスレッドの機械イメージは、スレッドのコンカレント状態をもたらす命令を捜し出し置き換えるようにインストルメントされる。複数のスレッドが実行されている間に、コンカレント状態情報はメモリに記録される。この記録されたコンカレント状態情報は、分析され、一致しない動的コンカレント状態遷移が、コンカレントエラーとして報告される。



1

【特許請求の範囲】

【請求項１】　プログラムにおけるコンカレントエラーを検出するためのコンピュータによって実施される方法であって、プログラムの複数のスレッドの機械語をインストルメンティングして、該スレッドのコンカレント状態をもたらす命令を捜し出して置き換え、前記複数のスレッドが実行されている間にメモリにコンカレント状態情報を記録し、該記録されたコンカレント状態情報を分析し、一致しない動的コンカレント状態遷移をコンカレントエラーとして報告することを特徴とする方法。

【請求項２】　ロックを獲得し解放する命令を捜し出し、該捜し出された命令を実行フローをインタセプトする命令と置き換え、実行されている間に前記複数のスレッドがロックを獲得し解放するオーダーを記録し、前記複数のロックがロックを獲得するオーダーにおいて一致しないコンカレント状態遷移をサイクルとして報告し、実行フローをインタセプトするときにモニタリングプロシージャをコールすることを含む請求項１記載の方法。

【請求項３】　各スレッドに対して、スレッドに関連した現在のロックセットを維持し、ロックが獲得されるときに前記ロックセットにそのロックを加え、前記ロックが解放されるときに前記ロックセットからそのロックを削除し、前記ロックが獲得されるオーダーをロックログに記憶させ、新しいロックが獲得されるときに前記ロックセットにロックセットペアを記憶させ、該ペアは、前記ロックセットの現在のメンバーと、前記ロックセットのメンバーでない前記新しいロックとの各独自の組み合わせであり、各ペアに対して、前記ロックセットの現在のメンバーが第１の位置に記録され、前記新しいロックが第２の位置に記録されるか、または、前記ロックセットの現在のメンバーが第２の位置に記録され、前記新しいロックが第１の位置に記憶されるか、するように決定されたオーダーにて前記ペアを記憶させることを含む請求項１記載の方法。

【請求項４】　前記ロックログのペアは、順序付けされたロックグラフとして表されており、さらに、各ロックのためのノードと、第１のロックを第２のロックに接続する方向付けエッジとを備えており、該エッジの方向は、前記第１および第２のロックが獲得されたオーダーを示しており、第１のノードおよび第２のノードが反対方向に向けられた第１のエッジおよび第２のエッジによって接続されるようなトポロジカルソートを前記ロックグラフに対して行い、前記第１および第２のノードに関連したロックをコンカレントロッキングエラーとして報告することを含む請求項３記載の方法。

【請求項５】　ヒープの各アドレスに対して、メモリアロケーションおよびメモリデアロケーション命令を捜し出し置き換えることによってシャドーヒープにおけるシャドーアドレスを維持し、前記ヒープのアドレスのデータにアクセスする命令を捜し出し、該アクセス命令をモニタリングプロシージャに対する命令で置き換え、前記モニタリングプロシージャによって対応するシャドーアドレスにロックセット情報を記録し、前記アクセス命令によるアクセスが前記記録されたロックセット情報と一致しない場合に、一致しないコンカレント状態をデータ競合状態として報告することを含む請求項１記載の方法。

2

【請求項６】　前記ヒープのアロケートされたアドレスへのアクセスを検出することを含む請求項５記載の方法。

【請求項７】　現在アクセスしているスレッドの現在のロックセット情報を前記ヒープのアドレスにアクセスしている前のスレッドの記憶されたロックセット情報でインターセクトし、前記セットのインターセクションが空である場合に、データ競合状態を報告し、現在のロックセットと前のロックセットとが異なる場合に、現在のロックセット情報と記憶されたロックセット情報とのインターセクションを記憶し、前記アクセスが前記アドレスへの書き込みである場合に、前記現在のロックセットと記憶されたロックセットとのインターセクションを書き込むことを含む請求項６記載の方法。

【請求項８】　前記スレッドアクセスが共用されないときに、アクセスするスレッドのアイデンティティを記憶することを含む請求項５記載の方法。

【請求項９】　読み取りアクセスに対して読み取りロックセットを維持し、書き込みアクセスに対して書き込みロックセットを維持し、読み取りアクセスに対して前記読み取りロックセットに読み取りロックを加え、書き込みアクセスに対して前記書き込みロックセットに書き込みロックを加えることを含む請求項１記載の方法。

【請求項１０】　前記ロックセットをアレーのエントリーに記録し、特定のエントリーに記憶されたロックセットに関連した各シャドーアドレスでのアレーのその特定のエントリーに対するポインターを記憶させ、特定のロックセットからハッシュ値を決定し、該ハッシュ値によって決定されるようなハッシュテーブルに前記ポインターを記憶させることを含む請求項５記載の方法。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、一般的には、コンピュータシステムにおいてプログラムスレッドをコンカレントに実行することに関し、より詳細に述べるならば、コンカレントエラーを検出することに関するものである。

【０００２】

【従来の技術】コンプレックスコンピュータシステムにおいては、処理をスピードアップするために、多重実行パスまたは“スレッド”が使用されうる。例えば、ユーザとの対話は、フォアグラウンドスレッドによって取り扱われうるが、一方、データベース更新の如き計算集中

オペレーションは、バックグランドスレッドによって取り扱われうる。また、性能を改善するために、パラレルスレッドが多重プロセッサにて実行されうる。

【０００３】コンカレントエラーを検出することが問題である。一つの型のコンカレントエラーは、“デッドロック”、すなわち、デッドロック状態である。例えば、スレッドＡが共用リソースＸを所有し、また、リソースＹを獲得することを必要とするような場合に、デッドロック状態が存在する。リソースＹは、リソースＸを獲得することを必要としている第２のスレッドＢによって所有されている。この結果、リソースのコンフリクトが生ずる。典型的には、デッドロック状態により、システムの意図された機能が停止させられてしまう。

【０００４】もう一つ別の型のコンカレンシー問題は、競合状態である。データ状態がスレッド実行のスピードおよびオーダーに依存する場合に、競合状態が存在しうる。例えば、あるスレッドは、他のスレッドによって書き込まれるアドレスのデータを読み出す必要があるかもしれない。読み出される実際のデータは、個々のスレッドにより読み取りおよび書き込みのオーダーに依存する。このような競合状態から生ずる非決定論により、そのプログラムはエラーを含む結果を生じてしまう。

【０００５】知られている従来の技術は、コンカレントエラーを静的にチェックするものである。典型的には、従来の技術は、プログラムが正しい、すなわち、“真”であるという定理を立証している。

【０００６】厳密なロッキングオーダーまたはデータアクセスシーケンスは、スレッドが実行されている間に、動的に変化することがあるということが問題である。このような変化は、静的チェッカーでは検出し得ない。したがって、コンカレントエラーを動的にチェックすることが望まれている。

【０００７】

【発明の概要】本発明によれば、プログラムにおけるコンカレントエラーを検出するためのコンピュータによって実施される方法が提供される。多重プログラムスレッドの機械語は、スレッドのコンカレント状態をもたらす命令を捜し出して置き換えるようにインストルメントされる。置換命令は、モニターのプロシージャをコールする。

【０００８】本発明は、上位概念において、本特許請求の範囲の請求項１に限定されたような、プログラムにおけるコンカレントエラーを検出するためのコンピュータによって実施される方法にある。

【０００９】後述するように、モニタープロシージャは、多重スレッドが実行されている間、メモリにコンカレント状態情報を記録する。その記録されたコンカレント状態情報は、分析され得て、一致しないコンカレント状態遷移は、コンカレントエラーとして報告され得る。

【００１０】置換される命令は、ロックを獲得し解放する命令であるのが好ましい。この場合において、記録された状態情報は、ロックのロッキングオーダーである。ロッキングオーダーにおけるサイクルは、デッドロック状態を示している。

【００１１】ヒープのアドレス毎に、シャドーアドレスが、シャドーヒープに維持される。これは、メモリアロケーションおよびデアロケーション命令を捜し出し置き換えることによって行われうる。この場合において、記録された状態情報は、データがアクセスされるオーダーに関連付けられている。一致しないアクセスオーダーは、データ競合状態として報告される。

【００１２】

【発明の実施の形態】次に、添付図面に基づいて、本発明の実施の形態について、本発明をより詳細に説明する。

【００１３】図１は、多重プログラムスレッドのコンカレントエラーを動的に検出するのに使用されうるプロセス１００を示す。プログラマーは、通常のプログラミング方法を使用してソースプログラム１１０を作成する。ソースプログラム１１０は、処理されるとき、コンピュータシステム（ＣＰＵ）１９０において同時に実行されるようになっている。コンピュータシステム１９０は、１つのメモリおよび１つのプロセッサを含むか、関連して使用される複数のメモリおよびプロセッサを含む。プログラム１１０は、コンパイラ１１１によってオブジェクトモジュール（thread obj）１２０へとコンパイルされる。

【００１４】ソースプログラム１１０が実行されている間そのソースプログラムの挙動に関して報告を行なうためのモニタープログラム１３０も作成される。ソースプログラム１１０の実行に関してモニタープログラム１３０が如何にして報告を行なうことができるかについての詳細については、後述する。モニタープログラム１３０は、コンパイラ１３１によってオブジェクトモジュール（mon obj）１２１へとコンパイルされる。リンカー１４０は、オブジェクトモジュール１２０－１２１を機械イメージ（.exe）１５０へとリンクする。機械イメージ１５０は、“バイナリー”プログラムまたは“テキスト”として知られている。スレッドの実行に関する報告をするためのイメージ１５０の統合部分は、“モニター”１５５と称される。

【００１５】ここで注意すべきことは、リンカー１４０は、コンパイラー１１１および１３１によって作成されたシンボルテーブル１４１にアクセスしうるということである。シンボルテーブル１４１を使用して、リンカー１４０は、例えば、オブジェクトモジュールの間で共通なプロシージャコールおよびデータレファレンスを分析しうる。また、ここで注意すべきことは、プログラムのコンパイリングおよびリンキングにより、プログラムのテキスト部分１５０と共に通される定数値を記憶するデ

ータセクション（data）１５１が発生されうるということである。一つの実施例として、プログラムテキストまたは機械イメージ１５０は、機械命令、例えば、オペランドおよびオペレータのみを含む。すべてのデータは、データセクション１５１に別々に記憶される。

【００１６】機械イメージ１５０は、インストルメンター１６０によって変更され、すなわち、“パッチ”される。インストルメンテーションは、イメージ１５０の選択された命令を捜し出し置き換えるプロセスである。使用されうるインストルメンターの例は、１９９６年７月３日にShrivastava 氏等に与えられた米国特許第５５３９９０７号「System for Monitoring Computer System Performance」に開示されている。

【００１７】各選択された命令は、１つまたはそれ以上の置換命令でオーバーライトされる。例えば、置換命令は、サブルーチンへの飛び越し命令（jsr）１７１でありうる。jsr 命令は、モニター１５０のプロシージャ１７３のうちの一つを“コール”（１７２）することができる。プロシージャ１７３は、選択された命令がまさに実行されようとしている時の、システムのコンカレント状態を調べ、記録し、分析するように構成されている。

【００１８】別の実施例において処理をスピードアップするために、コール１７２は、インライン命令として実施されうる。すなわち、インライン命令は、プロシージャ１７３を実施する。したがって、以下の説明においては、コール１７２は、呼び出されるルーチンのサブルーチンコールまたは等価なインライン命令を意味している。

【００１９】機械語１５０がインストルメントされた後、この機械語は、システムのメモリに記憶される。このために、インストルメンテーションは、“バイナリーリライト”として知られている。

【００２０】変形機械語１５０は、ローダー１８０によってプロセッサ１９０へロードされうる。プログラムを実行している間に、挿入された“コール”１７２は、スレッド実行をインタセプトする。プロシージャ１７３は、システムについての選択されたコンカレント状態情報１９５を記録する。記録された状態情報は、それら状態に関してプログラムの挙動を報告１９６するために分析されうる。

【００２１】例えば、一致しないオーダーにおけるロッキング、または、一致しないオーダーにおけるデータの記憶は、コンカレントエラーを反映していることがありうる。モニタープロシージャ１７３の完了後、置換された命令が実行でき、プログラムスレッドの通常の実行は、サブルーチンからの復帰命令１７４を介して再開されうる。コンカレントエラーは、プログラムが実行されている間、または、モニタリングの完了後に報告されうる。

【００２２】インストルメンテーション中に置き換えられる特定の命令１７１は、どのスレッドコンカレント状態を分析し報告すべきかに依存している。デッドロック状態を検出するために、ロックを獲得し解放させるのに使用されるプロシージャをコールするすべての機械命令がインストルメントされる。競合状態を検出するために、ロックを獲得し解放させるのに使用されるプロシージャをコールするすべての機械命令がインストルメントされ、それに加えて、データアクセス命令、例えば、ロードおよびストア、並びに、メモリアロケーションおよびデアロケーションコールもインストルメントされる。

【００２３】図２を参照して、デッドロックコンカレントエラーの検出について説明する。システム１９０において、複数のスレッド（Ａ、Ｂ、Ｃ）２１１－２１３が命令をコンカレントに実行している。スレッド２１１－２１３は、それらのアクティビィティー、例えば、共用リソースへのアクセスを同期するためロック（ａ、ｂ、・・・、ｅ、ｆ）へアクセスする。

【００２４】所定のアクティビィティーを同期するために、スレッドは、関連するロックを獲得するためにコール２２１をなす。そのロックをホールドすることにより、そのスレッドは、他のスレッドが同じ共用リソースを使用しないようにすることができる。そのアクティビィティーが完了したときに、コール２２２がそのロックを解放する。単一のコールが複数のロックを獲得し、または解放することもありうる。マニピュレートされる特定のロックは、コール２２１－２２２のアーギュメントとして通されうる。もし、所望のロックが使用できない場合、例えば、そのロックが他のスレッドによってホールドされている場合には、その獲得しようとしているスレッドは、そのロックが使用できるようになるまで、待ち状態にホールドされうる。

【００２５】デンドロック状態を検出するための本発明の方法によれば、ロックが正しく獲得されるべきオーダーが予め規定される。これは、マルチプロセスおよびマルチスレッデッドプログラムにおけるロックの獲得をマネージするための普通のディシプリンである。反例として、スレッドＡは、ロックａをホールドし、スレッドＢは、ロックｂをホールドする。前に進むためには、スレッドＡもロックｂを必要とし、スレッドＢは、ロックａを必要とする。獲得オーダーが競合するこのような場合には、どちらのスレッドも前に進めない。これは、デッドロックのクラシカルなケースである。

【００２６】コンプレックス実行環境においては、処理アクティビィティーを同期するために数千のロックが使用されうる。一致したロッキング挙動は、ある程度までは、静的にモニターされうる。もし、ロックが獲得されるオーダーが特定の実行に依存している場合には、デッドロック状態を検出することは非常に難しいものとなる。何故ならば、ことなるロッキングオーダーにおける置換の数は、非常に大きくなるからである。

【００２７】したがって、図１のモニター１５５は、スレッドが実行されている間におけるロックが獲得され解放されるオーダーに関連したコンカレント状態情報を記憶するデータ構造を維持する。この状態情報は、プログラムが一般的な使用のために利用できるようにされる前に、ロッキングオーダーにおけるエラーを検出するために分析されうる。

【００２８】もし、インストルメンター１６０がマネージメントプロシージャをロックするすべてのコール１７１を捜し出しパッチする場合には、１つのロックが獲得され、または、解放される毎に、それらスレッドの実行フローは、図１のプロシージャ１７３の１つによってインタセプトされうる。

【００２９】より詳細に述べるならば、モニターは、各スレッドに対して１つのロックセット２３０を維持する。１つのロックが獲得されるときはいつでも、そのロックは、そのスレッドに関連したロックセット２３０のメンバーとなる。そのロックが解放されるとき、そのロックは、そのセットから削除される。例えば、もし、スレッドＡがロックａ、ｂ、ｃを獲得し、後でロックｂを解放する場合には、そのロックセットは、メンバー｛ａ、ｃ｝を有する。

【００３０】さらにまた、モニターは、ロックログ３００にすべてのロック獲得を記録する。図３に示されるように、獲得されたロックは、順序付けされたペア３０１－３１２として次のように記録される。例えば、スレッドＡは、順に、ロックａ、ｂ、ｃを獲得する。ロックｂが獲得されるとき、そのペア（ａ、ｂ）３０１は、そのログに記録される。これは、ロックｂがロックａの後で獲得されたことを示している。その後、スレッドがロックｃを獲得するとき、それらペア（ａ、ｃ）および（ｂ、ｃ）もまた記録される。何故ならば、ロックａおよびｂは、共に、ロックｃが獲得されたときでも、スレッドＡのロックセットのメンバーであるからである。

【００３１】そのプログラムの実行のモニタリング中にロッキングオーダーが変更されない場合には、そのオーダリング関係を利用する最適化が可能である。このような最適化において、１つのスレッドによってホールドされるロックのセットは、最も前に獲得されたものから、最も最近に獲得されたものまで、時間的順序にて維持される。このような最適化でもって、新しいロックがあるスレッドによって獲得されたときには、最も最近に獲得されたロックおよび今獲得されつつあるロックからなるペアがログに記憶される。

【００３２】したがって、一般的な場合において、あるスレッドがあるロックを獲得するときはいつでも、あるペアが、そのスレッドのロックセットの現在のメンバーとその新しく獲得されるロックとの組合せ毎に、ロックログ３００に記憶される。換言するばらば、それらペアは、ロッキングのオーダーを表している。もし、ロックログ３００が既に同一ペアを記録している場合には、その新しいペアは、記録されない。すなわち、そのログは、独自のペアのみを記憶する。ことなるオーダーにて獲得されたロックを表す２つのペア、例えば、（ａ、ｂ）と（ｂ、ａ）とは同一ではない。

【００３３】図３の例であるログ３００は、ロックａ、ｃおよびｄを獲得しているスレッドＢを示している。その後で、スレッドＡは、ロックａおよびｃを解放したが、ロックｄ、ｅおよびｆを獲得した。それから、スレッドＢは、ロックａ、ｃおよびｄを解放し、ロックｆおよびｂを獲得する。最後のロック獲得は、ログ３００においてペア（ｆ、ｂ）３１２として反映されている。

【００３４】図４は、図３のロックログ３００の図式的表示４００を示している。ここで、各ロックは、ロックオーダリンググラフ４００におけるノード４０１－４０３であり、ノードのペア間の方向付けエッジ４０４－４０５は、それらロックが獲得された相対的オーダーを示している。

【００３５】ここで、この例のロッキングオーダーによれば、反対方向における２つの相互接続エッジ４０４および４０５を有するノード４０２および４０３が生ずることは明らかである。これは、サイクルと称される。サイクルは、スレッドが競合オーダーにてロックを獲得していることを示している。方向付けグラフにおけるサイクルは、ロックログ３００のペアに関してトポロジカルソートを行うことによって検出されうる。

【００３６】他のアプリケーションのためのトポロジカルソートについては、D.Knuth 氏による「The of Programming, Volume I, Fundamental Algorithms, Addison-Wesley, 1968, pp.258-268」に一般的に開示されている。サイクルは、潜在的なデッロック状態の証拠であり、したがって、報告される。

【００３７】ここで注意すべきことは、ロックオーダリンググラフ４００にサイクルがないことは、そのプログラムが絶対的にデッドロックのないことを意味するものでないということである。何故ならば、他のリソース同期化機構がデッドロック状態に巻き込まれうるからである。さらに、スレッドが異なるオーダーにおいてロックを獲得するようにさせる入力データの別のセットもありうる。一般的に、コンカレントエラーを動的にチェックしても、そのプログラムの特定の実行についてスレッドが適切にロックすることを立証することができるだけである。

【００３８】別の型のコンカレントエラーは、データ競合状態である。競合状態は、複数のスレッドが同一アドレスのデータにアクセスするときに起こる。例えば、２つのスレッドの両者が変更する意図をもってデータを読み取る。この場合において、最終データ状態は、どちらのスレッドが最初に完了するかに依存している。

【００３９】大抵のオペレーティングシステムにおいて

は、スケジュラーは、任意のオーダーにおいてスレッドを実行する自由を有しており、同期化の制約のみを受ける。マルチプロセッサシステムにおいては、実行は、例えば、異なるプロセッサのキャッシュミスおよびその他のロード条件に依存した異なるスピードで行われうる。したがって、もし、完了時間が動的に変化する場合には、最終データ状態も変化しうる。

【００４０】もし、各共用データがロックで保護される場合には、データ競合は避けられうる。大きなプログラムの複数のスレッドの間では、どのロックがどのデータに関連しているのかを静的に確認することは難しい。

【００４１】図５は、アドレスレベルでデータ競合状態を検出するのに使用されうるデータ構造を示している。ランダムアクセスメモリ（ＲＡＭ）５００は、“ヒープ”５１０のアドレスにデータを記憶する。ヒープ５１０は、データを共用するために複数のプログラムスレッドによってコンカレントにアクセスされうる。

【００４２】データは、アドレス５１１の如き独自のアドレスに記憶される。これらアドレスは、仮想メモリアドレスでありうる。これは、ダイナミック物理メモリ（ＤＲＡＭ）の総量がアドレスされうるものよりも少ないことを意味している。１つのありうる仮想メモリシステムにおいては、アドレス５１１は、３２ビット値として表され得て、メモリの４ギガバイトのアドレッシングを可能とする。物理メモリとしては、そのうちの少量部分のみしか実施され得ない。ここで理解すべきことは、これは、多くの変化のうちの１つのみであり、例えば、６４ビットワードしか使用され得ないということである。

【００４３】ヒープ５１０のアドレス毎に、シャドーヒープ５２０もまた維持される。これは、アドレス５１１とアドレス５２１との間の１対１の対応５０１があることを意味している。シャドーヒープ５２０は、アクセスコンカレント状態情報を記憶するのに使用される。これは、ヒープのあるアドレスのデータがアクセスされるときはいつでも、アルゴリズムがそのように要求する場合には、現在のデータアクセスコンカレント情報は、シャドーヒープ５２０の対応するアドレスにて更新される。

【００４４】図１の機械語１５０は、次のように変更される。ヒープ（ロードおよびストア）のアドレスにアクセスするすべての命令１７１は、モニター１５５のプロシージャ１７３へのコール１７２によって置き換えられる。さらに、ヒープ５１０のアドレスをアロケートおよびデアロケートするプロシージャへの機械語１５０によるコールは、手順をモニタリングするコールで置き換えられる。この後のアクティビティーは、シャドーヒープ５２０がそのヒープ５１０のアドレスを正確に反映するようにするために行われる。

【００４５】図６は、シャドーヒープ５２０のアドレスの１つにおけるワード６００を示している。このワード６００は、２つのフィールド、すなわち、アクセス状態フィールド６１０と、ロックセット状態フィールド６２０とを有している。後述するように、ある特定の実施においては、状態フィールド６２０は、２つのサブフィールド６２１－６２２を含み得る。アクセス状態フィールド６１０は、１から３ビット、例えば、アクセス（Ａ）、共用（Ｓ）および読取り／書込み（Ｒ／Ｗ）６１１、６１２および６１３を有しうる。

【００４６】ワード６００のアドレスは、ヒープ５１０の対応するアドレスがアロケートされるとき、シャドーヒープ５２０にアロケートされうる。ワード６００は、すべてのビットを零へセットするように初期化されうる。これは、ヒープアドレスの対応するワードが決してアドレスされていないことを示している。ヒープにおけるワードのデアロケーション時には、そのシャドーヒープの対応するワードがデアロケートされ、それが以前に記憶していたどの情報も失われる。このデアロケーションは、ライブラリデアロケーションルーチンに対するプログラムにおける明示コールを通して起こりうるか、または、後述するように注釈されたガーベジコレクターの如き自動記憶リクラメーションルーチンを通して起こりうる。

【００４７】図７は、シャドーヒープ５２０を維持するのに使用されうるプロセス７００を示している。図２の任意のスレッド２１１がステップ７１０でまさにヒープ５１０のアドレスのデータにアクセスしようとするとき、ステップ７２０において、シャドーヒープ５２０の対応するアドレスがアロケートされていたかを決定する。シャドーヒープアドレスがないことにより、ステップ７２５においてヒープ５１０のアロケートされていない部分にアクセスする誤った試みが検出される。

【００４８】もし、妥当なシャドーヒープアドレスに記憶された値が零（ステップ７３０における以前のアクセスのない）である場合には、ステップ７３２において、状態フィールド６１０のアクセスビット７１１を論理“１”へセットし、ステップ７３５において、アクセスしているスレッドに関連した現在のロックセットをロックセットフィールド６２０へ書き込む。置換された命令を実行した後、そのスレッドの実行を再開する。ロックセットがどのようにしてシャドーヒープ５２０のアドレスに記憶されかについては、後で詳述する。後述するように実際には、ロックセット状態フィールド６２０は、そのロックセットが記憶されるアレー素子に対するポインタを記憶する。したがって、ロックセットの“書き込み”は、ワード６００のポインタの記憶を意味している。

【００４９】その後、同じアドレスに対する別のアクセスにより、アクセスビット７１１が零でないことが検出される。この場合において、アドレスにアクセスしているスレッドのロックセットとシャドーアドレスに記憶さ

れたロックセットとのインターセクションをとり、ステップ７４０において残りのロックセットをロックセットフィールド６２０に記憶する。もし、インターセクションをとった後ロックセットフィールド６２０がステップ７５０において空である場合、例えば、共通の未決定のロックがない場合には、ステップ７５５においてそのアドレスに対するデータ競合状態がありうるか、または、さもなければ、戻る。アドレスをデアロケートすることにより、ワード６００は、その初期のすべてゼロ値へとリセットさせられる。

【００５０】この方法は、データ競合状態を誤って合図する可能性がある。これらは、誤りアラームとして知られている。誤りアラームは、単一のスレッドのみがデータをアクセスしている場合に起こりうる。

【００５１】図８は、プロセス７００がどのようにしてより弁別しうるコンカレント検出を行うように修正されうるかを示している。この変形においては、もし、アクセスビット７１１が非零である場合に、付加的なチェックが共用ビット７１２に関してなされる（ステップ７６０）。共用ビット７１２が零である間、スレッド識別子が、シャドーアドレスのフィールドに記憶される。

【００５２】その後のアクセスにおいて、もし、共用ビット７１２がなおも零である場合には、アクセスしているスレッドの識別子が、シャドーアドレスに記憶された識別子と比較される。もし、それら識別子が異なる場合には、共用ビット７１２は、少なくとも２つのスレッドがそのアドレスにアクセスしたことを示すために、論理“１”にセットされる。

【００５３】ロックセットは、共用ビット７１２が“１”へセットされるまでは、シャドーアドレスに記憶されない。何故ならば、そのアドレスが１つより多いスレッドによって共用されるまでは、エラーチェックを行う必要がないからである。スレッド識別子およびロックセットは同時にシャドーアドレスに記憶される必要はないので、シャドーアドレスの同じフィールド６２０が、異なる時間にてこれらの値を記憶するのに使用されうる。

【００５４】共用ビット７１２がゼロのままである限り、そのプロセスは、ステップ７３５においてシャドーアドレスのロックセットフィールド６２０にそのスレッドのアイデンティティを単に書き込む。この場合において、ロックセットは、同じスレッドがそのアドレスのデータにアクセスし続ける限り、もはやインターセクトされない。もし、共用ビット７１１が“１”にセットされる場合には、空のセット７４０および７５０についてインターセクトし、チェックする。

【００５５】しかしながら、この方法は、あるアプリケーションに対しては粗過ぎて、なおも誤りアラームを生ずる可能性がある。例えば、多くのデータ処理システムにおいては、多くのスレッドがその後にこれらのアドレスを読み取ることがあるとしても、多くのアドレスが１つのスレッドによって一度初期化されるが、決して再び書き込まれない。この場合において、ロックは、それが最後に書き込まれたときに、そのアドレスを保護するために必要とされない。したがって、前述したコンカレントチェッキング方法は、多くの誤りアラームを報告する可能性がある。

【００５６】シャドーアドレスに記憶されたＲ／Ｗビット６１３は、この問題を克服するのに使用される。このビットは、共用ビット６１２がセットされるまでは、無視される。共用ビット６１２がセットされるとき、Ｒ／Ｗビット６１３は、次の書き込みアクセス時に“１”にセットされる。もし、共用ビット６１２が書き込みアクセスによりセットされる場合には、Ｒ／Ｗビット６１３もまたすぐにセットされる。

【００５７】シャドーアドレス６００に記憶されたロックセット６２０に対する更新は、Ｒ／Ｗビット６１３の状態とは関係なく、通常のように進められる。Ｒ／Ｗビット６１３の作用は、ビットが零であるときに、エラーメッセージを抑圧することである。何故ならば、少なくとも２つのスレッドがあるアドレスのデータにアクセスしておらず（共用ビット６１２は“１”にセットされる）且つそのアドレスの共用後にそれらスレッドの少なくとも一方によってそのアドレスが書き込まれていない（Ｒ／Ｗビットが“１”にセットされる）限り、アクセスエラーは起こり得ないからである。

【００５８】ある場合に使用されうる１つのプログラミング取決めは、１つより多いロックで単一のアドレスに記憶されるデータを保護する。この取決めにおいては、スレッドは、データが変更される前に、すべてのロックを獲得しなければならないが、１つのスレッドは、そのデータを読み取るために１つのロックを必要とするだけである。本発明の方法は、このようなプログラミング取決めを受け入れるように変更されうる。ある書き込みアクセス時に、そのプロセスは、前述したように、シャドーアドレスを更新しうる。ある読み取りアクセス時に、空セットに対するロックセットインターセクションおよびテストが前述したように行われる。しかしながら、この場合において、シャドーアドレスに記憶されたロックセットは、インターセクトされたセットで更新されない。

【００５９】このコンカレントエラー検出法は、ロックが２つのモード、すなわち、読み取りモードおよび書き込みモードのうちの一方において獲得されうるような単一ライター／多重リーダーロックを取り扱うように更に改良されうる。もし、あるスレッドが書き込みモードにおいてあるロックを獲得する場合には、そのスレッドは、他のスレッドがどのような場合でもロックをホールドしないように保証される。もし、あるスレッドが読み取りモードにおいてそのロックを獲得する場合には、他

のスレッドが書き込みモードにおいてロックをホールドしないことが保証されるだけである。

【００６０】単一ライター／多重リーダーロックにより、多重スレッドがあるアドレスに記憶されたデータをコンカレントに読み取れるようにされる。これは、多重ロックでアドレスを保護する概念に類似している。しかしながら、実施のための考え方は、本発明のコンカレントエラー検出法においてロックを受け入れる仕方のように、異なる。

【００６１】このような説明においては、単一のロックセットのみが１つのスレッドに関連付けられている。ロックがそのスレッドによって獲得され解放されるときに、ロックはそのセットに加えられ、そのセットから除かれる。単一ライター／多重リーダーロックを受け入れるために、２つのロックセット、すなわち、読み取りセットおよび書き込みセットが各スレッドに関連付けられる。

【００６２】ある書き込みロックが獲得されるとき、それは、両方のセットに加えられ、そのロックは、そのロックが解放されるときに、それら２つのセットから除かれる。読み取りロックの場合には、そのロックが読み取りセットに加えられ、読み取りセットから解放されるだけである。シャドーアドレスでの情報の処理は、読み取りアクセス中に読み取りセットのみが使用され、書き込みアクセス中に書き込みセットが使用されること以外は、前述したものと同様である。

【００６３】次に、ロックセットおよびロックセットに対するポインターをオルガナイジングするための好ましい仕方についてより詳細に説明する。図９は、複数のロックセット９０１－９０７を含むアレー９００を示している。このアレー９００における各エントリーは、ロックの独自のセットである。実際には、異なるロックセットの数は、小さく、したがって、テーブル９００は、大抵の場合において、メインメモリに記憶されうる。テーブル９００のエントリーは、特定のモニタリングセッション中において恒久的なものである。これは、それらエントリーが変更されず、テーブル９００から削除されないことを意味している。

【００６４】識別子として使用される小さな整数値が各ロックセットに関連付けられる。この識別子は、アレー９００へのインデックスである。テーブル９００におけるエントリーは、エントリー“０”から始まって順次、エントリー“１”、・・・等々の如く発生されていく。アレーへのインデックスは、そのエントリーの識別子である。図６のロックセット状態フィールド６２０に実際に記憶されるのは、これらの識別子である。

【００６５】テーブルを小さく保つために、一つのエントリーは、１回より多くはアレー９００に加えられない。したがって、補助ハッシュテーブル１１００が維持される。一つのロックセットエントリーがアレー９００

に発生される毎に、ハッシュテーブルエントリー１１１０も発生される。ハッシュキー値は、ハッシングファンクションをロックセットに直接に適用することによって導出される。ロックセット識別子（アレーインデックス）は、対応するハッシュテーブル値に記憶される。したがって、オペレーション中、ロックセットは、独自に識別されうる。

【００６６】デッドロックを検出する方法の場合のように、競合状態を検出するためのこれらの方法の使用により発生される結果は、モニターされる特定のプログラム実行に依存している。異なる入力データまたは事象のわずかに異なるシーケンスにより、プログラムを通しての実行パスが異なってくることがある。このために、この方法を使用するときには、起こりうる競合状態のすべてが検出されるというわけにはいかないことがありうる。実際に同期化がロッキング以外のあるメカニズムにより起きたときや、競合状態が実際に起きえないときに、潜在的競合状態が報告されることがありうる。

【００６７】さらに改善したものとして、チェックされるべきプログラムに注釈を付けることができる。プログラム注釈は、プログラマーの意図を示すためにそのプログラムに付される“シグナル”である。注釈は、プログラムの実行に影響を与えないものである。一つの効果として、注釈は、コンカレントエラー検出プロセスを改良するために実行中にチェックされうる。

【００６８】ソース注釈の例としては、所定のアーギュメントでの“ナル”プロシージャへのコールの形でありうる。このナルプロシージャは、その名が意味するように、正規の実行フローへの戻りにほかならない。ナルプロシージャへのコールは、インストルメンター１６０によって識別されうる。これらのコールは、アーギュメントを解釈するモニター１５５のプロシージャ１７３へのコールによって置き換えられうる。別の仕方として、注釈は、モニターのプロシージャへのコールの形でもありうる。これらのプロシージャは、シンボルテーブル１４１にて識別されうる。

【００６９】例えば、ある注釈は、次のアロケーションが読み取り命令によって共用モードにてアクセスされうるだけのものであるメモリアドレスに対するものであることを示すことがありうる。別の注釈は、アロケートされるメモリがそのスレッドに対して専用であることを示すことができる。明らかなように、このような注釈は、誤りアラームの数を減少させうるものである。

【００７０】デッドロック検出の場合に加えられうる注釈の別の例として、プログラムに使用されるロッキングオーダーが変更されていることを示す注釈がある。例えば、このような注釈は、ロッキングオーダーにおけるロックの位置が変更され且つそれを含むすべてのペアがログから除去されるべきであったことを示すことがありうる。

15

【００７１】動的競合検出の場合において、もし、プログラムが自動記憶リクラメーション（ガーベジコレクション）を与える場合には、モニターに対するコールの形の注釈は、記憶のあるレンジがリクラメートされており、シャドーヒープからデアロケートされるべきであることを示すために、プログラムのソースに含まれうる。

【００７２】本発明の特定の実施例について説明してきたのであるが、これらの実施例の種々な変形態様が考えられ、これら変形態様においても、前述したような効果のすべて、またはそれら効果のうちのいくつかを達成しうることは明らかであろう。したがって、本請求の範囲の記載は、本発明の範囲内に入るような変形態様のすべてを包含しようとするものである。

【図面の簡単な説明】

【図１】プログラムにおける同期エラーを動的に検出するためのプロセスを示すブロック図である。

【図２】多重スレッドおよびロックセットを示すブロック図である。

【図３】ロックログを示すブロック図である。

【図４】ロックオーダーリンググラフを示すブロック図である。

【図５】ヒープおよびシャドーヒープを示すブロック図である。

【図６】ロックセットを示すブロック図である。

【図７】コンカレントアクセスエラーをチェックするためのプロセスを示すブロック図である。

16

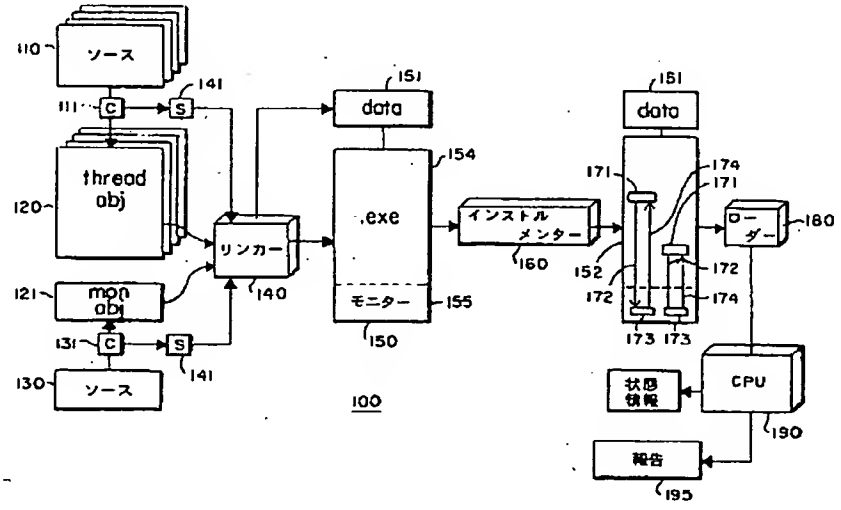
【図８】コンカレント共用アクセスエラーをチェックするための変形フローを示す図である。

【図９】ロックセットアレーおよびハッシュテーブルを示すブロック図である。

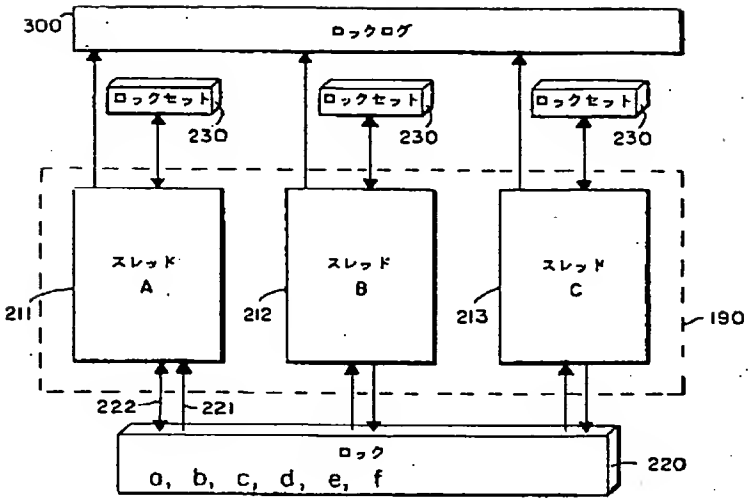
【符号の説明】

１００　プロセス
１１０　ソースプログラム
１１１　コンパイラ
１２０　オブジェクトモジュール
１２１　オブジェクトモジュール
１３０　モニタープログラム
１３１　コンパイラ
１４０　リンカー
１４１　シンボルテーブル
１５０　機械イメージ
１５１　データセクション
１５５　モニター
１６０　インストルメンター
１７１　サブルーチンへの飛び越し命令
１７２　コール
１７３　プロシージャ
１７４　サブルーチンからの復帰命令
１８０　ローダー
１９０　プロセッサ
１９５　コンカレント状態情報

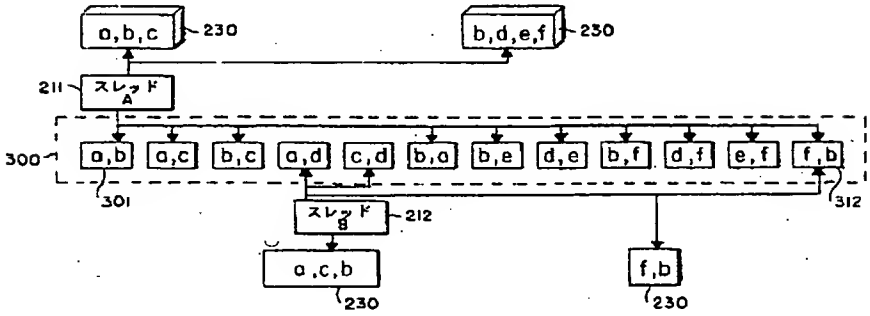
【図１】



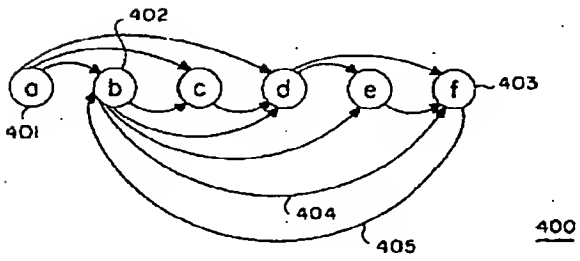
【図２】


300
ロックログ
ロックセット
230
ロックセット
230
ロックセット
230
スレッド
A
211
スレッド
B
212
スレッド
C
213
190
222
221
ロック
a, b, c, d, e, f
220

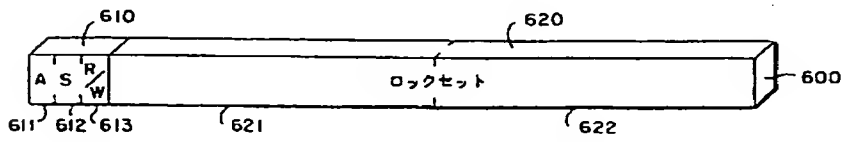
【図３】


a,b,c
230
b,d,e,f
230
211
スレッド
A
300
a,b
a,c
b,c
a,d
c,d
b,a
b,e
d,e
b,f
d,f
e,f
f,b
301
312
スレッド
B
212
a,c,b
230
f,b
230

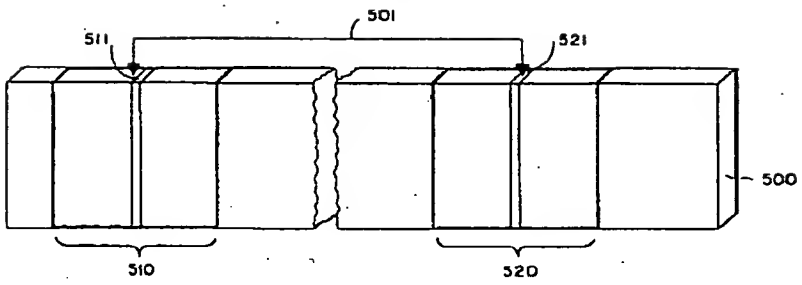
【図４】


402
a
b
c
d
e
f
403
401
404
405
400

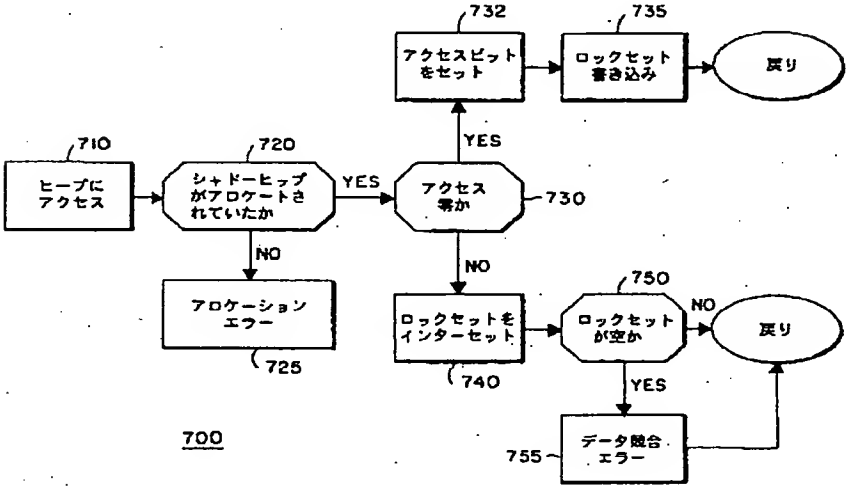
【図６】


610
620
A
S
R
W
ロックセット
600
611
612
613
621
622

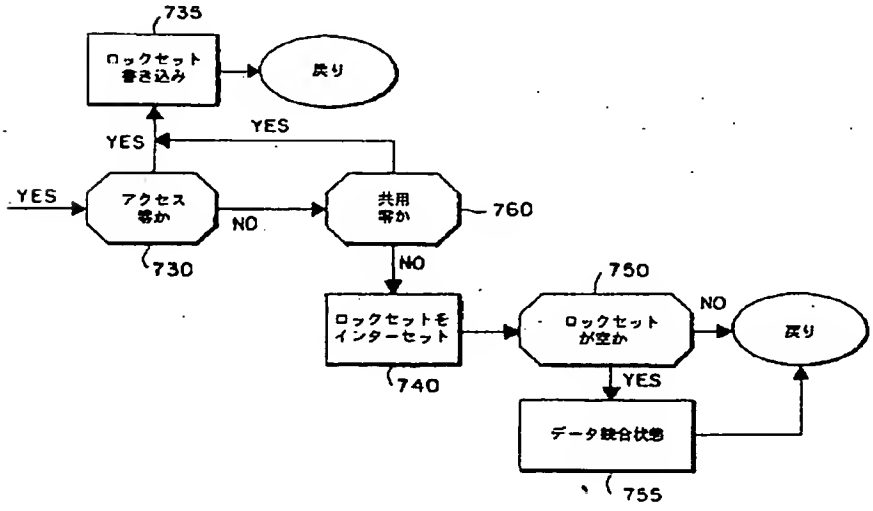
【図５】


501
511
521
500
510
520

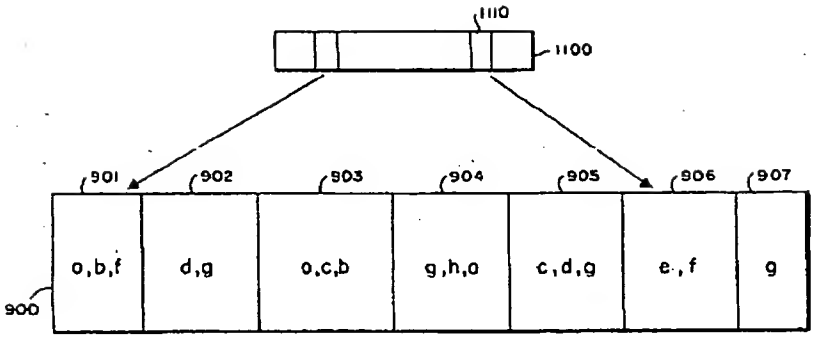
【図７】


732
735
アクセスビットをセット
ロックセット書き込み
戻り
710
720
ヒープにアクセス
シャドーヒープがアロケートされていたか
YES
アクセス等か
730
YES
NO
NO
アロケーションエラー
725
ロックセットをインターセット
740
750
ロックセットが空か
NO
戻り
YES
データ競合エラー
755
700

【図８】


735
ロックセット書き込み
戻り
YES
YES
YES
アクセス等か
730
NO
共用等か
760
NO
ロックセットをインターセット
740
750
ロックセットが空か
NO
戻り
YES
データ競合状態
755

【図９】



フロントページの続き

(72)発明者　チャールズ　ジー　ネルソン
アメリカ合衆国　カリフォルニア州　94306　パロ　アルト　カレッジ　アベニュー　770

(72)発明者　ステファン　サヴェージ
アメリカ合衆国　ワシントン州　98052　レッドモンド　ノースイースト　ワンハンドレッズ　コート　17831

(72)発明者　パトリック　ジー　ソバルヴァーロ
アメリカ合衆国　カリフォルニア州　94301　パロ　アルト　エマーソン　ストリート　312